

Computer Science Department

TECHNICAL REPORT

"Preliminary Studies for a First Order
Logic of Knowledge and Action"

L. Morgenstern

Technical Report #262

December 1986

NEW YORK UNIVERSITY



Department of Computer Science
Courant Institute of Mathematical Sciences

251 MERCER STREET, NEW YORK, N.Y. 10012

NYU COMPSCI TR-262 c.1
Morgenstern, Leora
Preliminary studies for a
first order logic of
knowledge and action



**"Preliminary Studies for a First Order
Logic of Knowledge and Action"**

L. Morgenstern

Technical Report #262

December 1986

Preliminary Studies for a First Order Logic of Knowledge and Action

Leora Morgenstern

New York University
Department of Computer Science
New York, N.Y. 10012

ABSTRACT

AI planners which are designed for robust domains must be able to work well with incomplete knowledge. In particular, they must be able to reason about what they need to know in order to perform an action, and how to get the information that they need. This paper presents a formalism which is expressive enough to describe these reasoning processes. We demonstrate that for our purposes, a first order logic of knowledge is superior to a modal logic of knowledge. Next, we describe the paradoxes that result from a first-order approach, and show how we can resolve them. Finally, we present a simple model of action and propose a solution to the knowledge preconditions problem based on our integrated theory of knowledge and action.

December 11, 1986

Preliminary Studies for a First Order Logic of Knowledge and Action

Leora Morgenstern

1. Introduction

Most AI planners work on the assumption that they have complete knowledge of their problem domain and situation. Standard AI planning programs, such as STRIPS [Fikes and Nilsson 1971] and NOAH [Sacerdoti 1977], are typically presented with a complete list of actions, their preconditions, and effects, so that formulating a plan reduces to searching through this pre-packaged list of action operators for a detailed action sequence that will achieve some desired goal. While this model of planning is conceptually simple and easy to implement, it is not adequate as an account of real world complex planning. In actual planning situations we often don't have enough information when we start out to draw up completely detailed plans. This is true of even simple plans: Robert Moore has pointed out [Moore 1980] that an agent may not even be able to specify completely the steps of a plan to call a friend. Unless he already knows the number by heart, he must first look it up in the phone book, so that the plan becomes:

- 1) Look up the friend's number, X .
- 2) Dial X on the telephone

Because he does not know what X is when he starts out, step 2) is not fully specified until step 1) is completed.

Incomplete knowledge plagues most of our real life planning situations, but we are nonetheless able to develop and successfully execute complex plans. The trick lies in our ability to draw up sketchy plans and figure out ways of making these plans concrete. We can reason about whether or not we have enough information to do a particular action; if we don't, we can plan to obtain the necessary information, either by looking it up or by asking a friend. Or we can decide to delegate an entire subplan to another agent, if he knows more about the particular task than we do. Knowledge acquisition and delegating responsibility are integral parts of our planning processes.

A robust planning system must have similar capabilities if it is to work in a reasonably complex domain. In particular, it needs a solid theory of knowledge, action, and communication, so that it can produce a coherent plan even when its knowledge of the problem domain and problem situation is incomplete. Some work has already been done in this field, most notably by Robert Moore [Moore 1980]. Moore has developed a theory of knowledge and action in which it is possible, for example, for an agent who knows the combination of a safe to reason that he knows how to open the safe. Agents in Moore's system are also capable of reasoning about action sequences, so that an agent can reason that he can do the sequence of (read combination of safe; open safe) even if he doesn't know the combination of the safe at the time of his reasoning.

Douglas Appelt [Appelt 1982] has extended Moore's system to a multi-agent system in which agents cooperate and share information to accomplish a single task. In Appelt's

system, an agent A can plan to tell another agent B a piece of information that B needs to carry out a particular task.

Due to the inherent limitations of Moore's formalism, however, neither Moore nor Appelt can attack the knowledge, action, and communication problem in its full generality. There is no way, for example, for Moore or Appelt to describe John's plan to bake a cake if he himself doesn't know how to use an electric mixer, but does know that Bill knows how to use one.

This paper presents a logic that is expressive enough to deal with the problems that faced Moore's system. In the next section of this paper, I review Moore's work on knowledge and action, and demonstrate its logical limitations. Afterwards, I present an alternative approach, discuss its power, and show how it can be integrated with Moore's work on knowledge and action. Finally, I briefly discuss how my logic of knowledge and action can be extended to a robust logic of knowledge, action, and communication.

2. Moore's Logic of Knowledge and Action

2.1. Some Historical Background and Motivation

Moore's work is motivated by the desire to formalize the interaction between knowledge and action. He has two central concerns:

- 1- explaining how an agent can decide that he knows enough to carry out an action; i.e., how knowledge affects action, and
- 2- showing how performing an action can change what an agent knows; i.e., how action affects knowledge.

Of the two problems, the first proves to be the more difficult, and as such provides the focus for much of Moore's research. Prior to his work, most AI planning programs, as we have mentioned, had glossed over the problem and simply assumed complete knowledge of problem domain and situation for all agents. There was thus no need at all for an agent to decide whether he knew how to do a particular action. He always did.

The first tentative attempts at a theory of knowledge and action came from McCarthy and Hayes in their landmark article 'Some Philosophical Problems from the Standpoint of Artificial Intelligence.' [McCarthy and Hayes 1969] There, McCarthy and Hayes argue that a planning program does need to explicitly reason about its ability to perform an action. For example, if an agent is forming a plan to call another agent on the telephone, he must realize that he needs to know the second agent's telephone number. Similarly, if an agent is trying to open a combination safe, he needs to know the combination of the safe.

McCarthy and Hayes did not attempt to formalize any general principles of knowledge. Rather, they explicitly wrote down knowledge precondition axioms for each action: to call a friend, one must know his telephone number, to open a safe, one must know its combination, and so on. Unfortunately, there are two major problems with this approach. In the first place, we note that knowledge precondition axioms are needed for nearly every action. This leads to an explosion of axioms in any robust planning system, which in turn leads to unacceptably long and slow proofs. Secondly, knowledge precondition axioms are unsatisfactory in that they have no *explanatory* power. That is, the axioms do not explain how we come to know how to perform an action. There seems, for example, to be a strong connection between what an agent needs to know in order to call someone on the phone, and what that agent needs to know in order to open a combination lock. Moreover, the sort of knowledge needed in these cases seems to differ markedly from the knowledge needed by an agent who wishes to drive a car. In general, however, knowledge precondition axioms take no account of these connections and similarities.

We will be referring to these problems under the umbrella heading of 'the problem of knowledge preconditions.' This is the problem that Moore's system seeks to solve. Rather than simply write down knowledge precondition axioms, Moore sought to develop a general theory of knowledge and action in which a few principles could characterize the knowledge that any agent needs to perform an action.

2.2. Review of Moore's Work

Traditionally, there have been two ways of dealing with the verb 'know' in standard logic. We can regard it as a modal operator that applies to propositions, or we can regard it as a syntactic predicate that ranges over names, or labels, of sentences. If we want to express the fact that John knows that snow is white, the first approach would lead to the expression $\text{Know}(\text{John}, \text{snow is white})$ while the second approach would lead to the expression $\text{Know}(\text{John}, \text{'snow is white'})$, since the string 'snow is white' can be considered the unique name of the sentence: snow is white. Moore adopts the first of these two approaches, basing his theory on the work of Jaakko Hintikka in the fields of knowledge and belief [Hintikka 1962, 1969].

There are various important properties of knowledge which any robust theory of knowledge must capture. Everything an agent knows must be true, agents should know the logical consequences of their knowledge, and so on. The following axiom schemata characterize an agent's knowledge in Moore's system.

M1: Axioms of ordinary propositional logic

M2: $\text{Know}(A, P) \Rightarrow P$

(Veridicality: Whatever is known is true)

M3: $\text{Know}(A, P) \Rightarrow \text{Know}(A, \text{Know}(A, P))$

(Positive Introspection: If an agent knows something, he knows that he knows it)

M4: $\text{Know}(A, (P \Rightarrow Q)) \Rightarrow (\text{Know}(A, P) \Rightarrow \text{Know}(A, Q))$

(Consequential Closure: Agents know all logical consequences of their knowledge)

M5: If P is an axiom, then $\text{Know}(A, P)$ is an axiom

(Necessitation)

A system defined by M1-M5 is equivalent to the modal logic S4 as defined in [Hughes and Cresswell 68], if we hold the knower fixed and substitute necessary for 'know.'

Moore, however, does not actually reason about knowledge and action within his modal framework. Rather than reason about the set of propositions that an agent knows, Moore instead reasons about what is true in the possible states of affairs, or 'possible worlds' that are compatible with an agent's knowledge. Recasting modal logic in terms of possible worlds is not a new notion in philosophy or logic. Historically, modal operators such as necessarily, possibly, and know, have been explained in terms of a possible world semantics. Three centuries ago, Leibniz defined a *necessary* statement to be one that was true in all possible worlds, a *possible* statement to be one that was true in at least one possible world.

More recently, Kripke has argued that concepts like necessarily and possibility can best be understood, not absolutely, but in terms of how they relate possible worlds [Kripke 1963a]. A world may be necessary relative to some possible worlds, but not to others. For example, a world in which Socrates dies is necessary relative to our world where all men are mortal, but might not be necessary relative to some world where people live forever. We can introduce an accessibility relation R that relates these possible worlds, so that $R(w_1, w_2)$ means that world w_2 is necessary relative to world w_1 . Kripke has in fact shown that by suitably restricting this accessibility relation, e.g. by positing that it is reflexive, symmetric and/or transitive, we can arrive at all of the standard systems of modal logic.

In much the same way, Moore (borrowing from [Hintikka 1969]) defines an accessibility relation for the modality 'Know', where $K(a, w_1, w_2)$ means that w_1 and w_2 are knowledge equivalent from a 's point of view. If, for example, John knows in W_0 that snow

is white, and w_1 is a possible world where snow is white, but w_2 is a possible world where snow is green, it is possible that $K(\text{John}, W_0, w_1)$, but it is not the case that $K(\text{John}, W_0, w_2)$ since W_0 and w_2 are not knowledge equivalent from John's point of view. If Fred, on the other hand, does not know in W_0 that snow is white and thinks that snow is green, then it may be possible that $K(\text{Fred}, W_0, w_2)$.

Moore introduces an axiom which allows us to translate back and forth between statements using 'Know' and those using 'K', and two axioms which posit the reflexivity and transitivity of K, and which thus capture the same characterizations on knowing as M1-M5. These axioms are presented, in slightly simplified form, below:

K1: Forall a, p, w_1 ($T(w_1, \text{Know}(a, p))$ iff Forall w_2 ($K(a, w_1, w_2) \Rightarrow T(w_2, p)$))

K2: Forall a, w_1 ($K(a, w_1, w_1)$)

K3: Forall a, w_1, w_2 ($K(a, w_1, w_2) \Rightarrow \text{Forall } w_3 (K(a, w_2, w_3) \Rightarrow K(a, w_1, w_3))$)

K1 says that if a knows that p in some world w_1 , then p is true in every world that is knowledge equivalent to w_1 from a 's point of view. Conversely, if p is true in every world that is knowledge equivalent to w_1 from a 's point of view, then a knows that p . It should be noted that this axiom implies that all agents know all facts that are true in all possible worlds, i.e. all axioms. This is because if a statement is true in all worlds, the right hand side of the if and only if statement is automatically true. **K2** says that **K** is a reflexive relation, and **K3** says that **K** is transitive.

Moore's reasons for his switch into possible world semantics were threefold. In the first place, he claims that proving theorems in terms of a first order theory of possible worlds is considerably more efficient than proving these same theorems within a modal logic. In the second place, Moore finds that the possible worlds ontology meshes nicely with the standard McCarthy-Hayes model of actions as functions on situations; he merely identifies possible worlds and situations. Thirdly, some of the ideas that philosophers have developed in connection with possible world theories play an important role in Moore's integrated theory of knowledge and action.

The most important of these notions is that of the 'rigid designator.' [Kripke 1972] Intuitively, a rigid designator designates the same individual in all possible worlds. For example, Franz Joseph Haydn is the rigid designator for the individual who we know was Kappellmeister to the Esterhazy family, composed the Farewell, Surprise, and Clock symphonies, and died in Vienna. But this rigid designator, Franz Joseph Haydn, would designate the same individual even in some possible world where he would not have composed the Surprise Symphony, or had died in London. A rigid designator fixes the reference of an individual in the actual world so that we can talk about it in other possible worlds. Typical example of rigid designators are names, such as Franz Joseph Haydn or Claude Monet, or numbers.

How does the idea of a rigid designator help Moore solve the problem of knowledge preconditions? According to Moore, an agent knows how to do an action if he knows of an executable description for that action. Moreover, we can in general say that we know who somebody is or what something is if we know a rigid designator for that person or thing. For example, we know what Bill's phone number is if we know that 460-7100 is his phone number, but not if we know that his phone number is Mary's phone number, because 460-7100 is a rigid designator, but 'Mary's phone number' is not. In particular, we know what entity an action refers to, and thus how to do an action, if we know of a rigid designator for that action. Since we also know how to do an action if we know of some executable description for that action, it follows that a rigid designator for an action must be an executable description for that action. Once we know of a rigid designator for an action, we know all that we need to know to actually perform that action.

In practice, how does an agent come to know a rigid designator for an action? Moore argues that many basic action-types such as Dial() and Move() can be thought of as rigid functions in the sense that they map rigid designators onto rigid designators. So as long as

we know rigid designators for the parameters of these basic action-types, we know a rigid designator for the action itself. To take a concrete example, we know how to do `Dial(phone-number(Bill))` if we know a rigid designator- or number - for Bill's phone number.

Other action types are more complex and cannot be thought of as rigid functions, but instead, as small procedures that are built up out of rigid functions. For example, `Puton(a,b)` can be defined as `(Movehand(Location(a)); Grasp; Movehand(Location(Top(b))); Ungrasp)`. `Puton(a,b)` is not a rigid function since `Location(a)` and `Location(Top(b))` can vary in different possible worlds. Nevertheless, this definition is assumed to be axiomatic, so that all agents know it. So again, knowing rigid designators for the parameters of the rigid functions implies knowing a rigid designator for the action.

Moore has thus characterized the knowledge preconditions for every single action with one basic principle. We can now prove, without adding further machinery, that John can open a safe if he knows a rigid designator for that safe's combination, that he can move an object to a table top if he knows rigid designators for the location of the object and table top, and so on. One axiom has replaced a potentially huge collection of knowledge precondition axioms.

2.3. Criticisms of Moore's System:

Unfortunately, Moore's system is beset by a host of problems. The first of these is the intuitive implausibility of the argument that an executable description of an action is a rigid designator for that action. Intuitively, a rigid designator for an individual seems to be that designation of the individual that stays fixed across all possible worlds. But it is hard to imagine that this is true of an executable description of an action. Take the action of dialing 460-7100 on the telephone, for example. In 1950, the executable description for that action was: Place one's finger in the 4th hole on the telephone dial and rotate clockwise, place one's finger in the 6th hole on the telephone dial and rotate clockwise, and so on. Today, an executable description for that same action might be: Place one's finger on the first button, 2nd row and push, place one's finger on the last button, 2nd row and push, and so on. And we can easily imagine a world where dial phones are obsolete, and the place- finger-in-hole-rotate sequence might be used to denote some completely different action.

Examples such as these would seem to indicate that an executable description cannot ever serve as a rigid designator for an action; that, rather, an executable description is a property of an action that varies among different possible worlds. Given any executable description of an action, we can imagine some possible world in which that executable description would not work. A possible response to this objection might be that we are using the word 'action' in too loose a sense. We might say, regarding the 1950-1986 telephone example, that dialing on a rotary phone and dialing on a push button phone are two completely different action-types, say `DialR()` and `DialPB()`. Or it is possible that we can think of there being just one `Dial` function that takes two arguments: a phone number and a phone type. Then the 1950's phone dialing would be an instance of `Dial(rotary,460-7100)` while today's phone dialing would be an instance of `Dial(push-button,460-7100)`, so that in effect, we have two different actions, each with its own rigid designator. But such a strategy strikes us as being forced and runs counter to our intuitive notions of what an action is. Our initial feeling of implausibility remains.

A second criticism of Moore's system is that his assumptions regarding an agent's knowledge seem excessively strong. In his system, all agents know all axioms. This is in general a dubious characterization of what agents in a realistic planning environment know and severely constrains Moore's theory of knowledge and action. In particular this assumption implies that all agents have the same level of procedural knowledge, so that all an agent might not know is some of the slot fillers in some action. But this seems wrong. Surely there is a difference between an agent who cannot call up Bill because he does not

know Bill's number, and an agent who cannot drive to Buffalo because he doesn't know how to drive a car. The lack of knowledge in the latter case seems to be more profound, for the agent doesn't even know the general procedure that he must follow in order to drive a car. He is not just missing a small piece of information.

Yet Moore's system admits of no such distinction. Any procedure to do an action, insists Moore, is known; either because it is a trivial procedure like Dial() or because it is defined as a sequence of primitive procedures by the axioms.

It is important to realize that it is impossible in Moore's system to relax the requirement that all axioms are known by all agents.^[1] It is not simply a matter of weakening M5; the possible worlds semantics on 'Know' are what entails this requirement, as is clear from examining K1, the axiom that allows us to translate between statements using 'Know' and those using 'K'. Any modal logic with a possible worlds semantics will suffer from this problem.

The third, and perhaps most important criticism of Moore's system is likewise a general criticism of any standard first order modal logic of knowledge: it is simply not expressive enough to handle a lot of our statements concerning knowledge. Moore himself has pointed out that there is no way to formalize in first order modal logic a statement such as 'John knows what Bill said' without specifying what it is that Bill said. This is because we cannot quantify over sentences in first order modal logic. Thus, we cannot formulate such sentences as 'John doesn't know what Bill said' or 'John knows that Bill knows something that he doesn't know.' Assuming that knowledge about actions is in the form of statements, we also cannot express 'John knows that Bill knows how to fire a gun' unless John himself knows how to fire a gun. Of course, in Moore's system, such a situation would be meaningless anyway, since all basic facts about actions are known, but we wish to develop a logic in which it *does* make sense for agents to learn procedures for action from other agents.

This lack of expressiveness of Moore's system seems to be the most powerful argument for an alternate system. Even if we ignore the first two arguments above, and agree that Moore presents the ultimate solution to the problem of knowledge preconditions, it is a hollow victory if his system cannot be extended to one that can express these very basic ideas about knowledge.

3. The Alternate Approach: A First Order Logic

3.1. 'Know' as a syntactic predicate: The Montague-Kaplan Paradox

What we want, then, is to construct a theory of knowledge that captures as closely as possible Moore's solution to the problem of knowledge preconditions, but manages to avoid the pitfalls outlined above. One possible strategy might be to turn to a higher order modal logic [e.g. Gallin 1975]; such a system would be capable of quantifying over propositions and would thus be expressive enough for our needs. Such an approach, however, would be unattractive for a number of reasons. In the first place, we would no longer have a complete proof procedure. Moreover, a higher order modal logic with possible worlds semantics still entails that all agents know all axioms, which was one of our primary objections to Moore's system.

We thus turn to the alternate approach that we mentioned at the beginning of the last section: treating 'know' as a syntactic predicate that ranges over names of sentences. Such an

1- It might seem possible, however, to define a class of statements that are not axioms and so are not necessarily known by all agents, but that remain true in all worlds that are physically accessible from the current world. There are various difficulties with such a scheme; at any rate, there are more severe problems with Moore's system as we shall see, so this doesn't seem to be a very worthwhile undertaking.

approach is very powerful. We can formulate all the examples of the last section that gave Moore's system so much trouble, and in a first order system as well. This is because we can quantify over names of sentences, which are just numbers, or strings, rather than quantifying over sentences themselves. So, 'John knows that snow is white' can be formulated as

$\text{Know}(\text{John}, \text{'snow is white'})$,

'John knows what Bill said' can be formulated as:

$(\text{Exists } x) (\text{Said}(\text{Bill}, x) \text{ and } \text{Know}(\text{John}, x))$,

'John knows that Bill knows something he doesn't know' can be formulated as

$\text{Know}(\text{John}, (\text{Exists } x)(\text{Know}(\text{Bill}, x) \text{ and } \neg \text{Know}(\text{John}, x)))$, and so on.

Despite its superficial appeal, however, treating 'know' as a syntactic predicate leads to severe difficulties. Richard Montague has shown, in fact, [Montague 1963] that any system that treats 'know' as a syntactic predicate and in which 'know' has some very general and desirable properties, must be inconsistent. [2] This result is a direct consequence of the Knower Paradox, which was first discussed by David Kaplan and Richard Montague in 1960 [Kaplan and Montague 1960]. The Knower Paradox, which can be regarded as a variant of the Liar Paradox, follows below:

If 'Know' is a syntactic predicate, we can construct, via Godel numbering, the following sentence:

(S) $\text{Know}(a, \text{'S'})$,

i.e.,

(1) $S \text{ iff } \text{Know}(a, \text{'S'})$

Suppose we have the following axioms on the predicate know:

(I) $\text{Know}(a, \text{'P'}) \Rightarrow P$ (whatever is known is true)

(II) $\text{Know}(a, \text{'P'}) \text{ and } \text{Logically-Derivable}(P, Q) \Rightarrow \text{Know}(a, \text{'Q'})$
 (if a knows P and Q is logically derivable from P then
 a knows Q; i.e., agents know logical consequences of
 their knowledge)

(III) $\text{Know}(a, \text{'I'}) \text{ and } \text{Know}(a, \text{'II'})$
 (agents know the basic principles of knowledge)

Is S true or false? If S is true, then by (1) a knows that not S. Therefore by (I), not S must be true, i.e. S is false. This is a contradiction, so S can't be true. Therefore S must be false. Since any agent who knows the basic logical axioms on knowledge can go through this reasoning, a can reproduce this argument (by (II) and (III)). So a knows that S is not true. But then by (1), S is true. Contradiction.

3.2. A Similar Problem- The Liar Paradox- and Tarski's Solution

The first thing we ought to notice- if there is to be any hope of developing a first order logic of knowledge- is the similarity between this argument and Tarski's argument that a classical logical language cannot contain its own truth predicate. Tarski's argument is based on the Liar Paradox: the apparent inconsistency of a statement such as 'This statement is a lie.' Tarski showed that if 'true' is regarded as a syntactic or arithmetic predicate, we can construct a sentence such as $P \text{ iff } \neg \text{True}(P)$, i.e.

(P) $\neg \text{True}(P)$

Obviously, inconsistency immediately follows if we allow such a statement.

2- This is true of any predicate having properties corresponding to those of veridicality, consequential closure, and necessitation.

To avoid the paradox, Tarski proposed that we do not have a single truth predicate in our language, but that instead there is a hierarchy of truth predicates $T_1, T_2, T_3 \dots$ corresponding to a hierarchy of languages $L_0, L_1, L_2 \dots$. At the bottom of the hierarchy we have L_0 , which is built up out of the usual operations of first order predicate logic on primitive predicates. Since such a language is incapable of containing a truth predicate, a metalanguage L_1 contains a truth predicate T_1 for L_0 . Of course the metalanguage L_1 cannot contain its own truth predicate, so a metalanguage L_2 contains a truth predicate T_2 for L_1 . This process can obviously be extended indefinitely.

According to Tarski, we implicitly use these different truth predicates when we speak. For example, when we say that 'It is true that 'snow is white' is true,' we are really saying:

'It is $true_2$ that 'snow is white' is $true_1$.'

The advantage of Tarski's hierarchy of truth predicates is that it completely avoids the Liar Paradox. Since L_0 does not contain its own truth predicate, we cannot create a sentence such as (P). 'True' simply isn't an available predicate at that point.

We might think it worthwhile to use Tarski's hierarchical approach to the predicate 'know', constructing a hierarchy of know predicates: $know_1, know_2, know_3 \dots$, and insist that $know_1$ first be introduced in L_0 's metalanguage. Such an approach would probably work as well as the hierarchy of truth predicates, and has in fact been suggested by Kurt Konolige in his theory of knowledge and action [Konolige 1982]. It turns out, however, that there are some serious difficulties with Tarski's strategy. Kripke discusses these problems [Kripke 1975] and also proposes an alternate solution to the Liar Paradox. Our strategy, in the next few sections, will be to study Kripke's solution to the Liar Paradox and see how we can adapt it to a solution to the Knower Paradox.

3.3. Kripke's Alternate Approach to Truth

Perhaps the most obvious objection to Tarski's truth predicate hierarchy is the fact that most people don't seem to implicitly attach any subscript when they use the word 'true.' Often, we don't even know which subscript we should attach. Suppose, for example, that I utter the sentence:

(2) Everything Nixon says is true

If Nixon has only made utterances that do not involve truth- i.e., that are in L_0 , then the applicable predicate is $true_1$. But presumably, Nixon has made utterances like 'I never tell a lie' and 'The truth is that everything McGovern said is a lie.' The fact is that we don't *know* the highest level of Nixon's utterances, so we *can't* assign a truth predicate to the occurrence of 'true' in (2). Nevertheless, there are many circumstances under which we would agree that (2) is a perfectly reasonable sentence.

Moreover, there are certain sets of sentences which cannot be analyzed by Tarski's approach irrespective of our knowledge. Suppose that John Dean says:

(3) All of Nixon's utterances about Watergate are false

and that Nixon says

(4) Everything Dean says about Watergate is false.

Clearly, Dean includes (4) in (3)'s scope, and Nixon includes (3) in (4)'s scope. By Tarski's theory, a sentence can only refer to the truth or falsity of a sentence with a lower 'level', so the Nixon- Dean situation would be impossible. Yet there are many situations where these sentences seem reasonable and can be assigned unambiguous truth values; e.g., if both Dean and Nixon have made some trivial but true statement about Watergate like 'The Republicans broke into the Democratic party headquarters in the Watergate,' both (3) and (4) will be false. Moreover, even in situations where these sentences are not reasonable- for example, in cases where Nixon and Dean say nothing but (3) and (4)- we would still not want to exclude

them from our language. The fact is that people do utter sentences like these and a language which cannot capture these utterances would be severely lacking in expressivity.

These and other problems have led Kripke to develop a new theory of truth that avoids both the Liar Paradox and Tarski's truth predicate hierarchy. The basic idea behind Kripke's theory is that not every sentence gets a truth value. Essentially, there are two types of sentences, the grounded sentences and the ungrounded sentences. Grounded sentences get a value of either true or false, while ungrounded sentences get an undefined truth value, the third value in a three valued logic.

The three valued logic that Kripke uses was originally developed by S.C. Kleene in order to formalize the way people reason with statements of indeterminate truth value [Kleene 1955]. Kleene suggested that people often mentally tag statements as true, false, or unknown, and that they systematically assign one of these 'truth values' to compound statements which are built up out of simpler statements. For example, if I tag 'Reagan is president' as true and 'SueEllen shot J.R.' as unknown, then I will tag the compound statement 'Reagan is president or SueEllen shot J.R.' as true, but I will tag the compound statement 'Reagan is president and SueEllen shot J.R.' as unknown.

Kleene proposes the following truth tables for the logical connectives $\vee$ and $\neg$:

$\vee$ t f u	$\neg$ t f u
-----	-----
t t t t	f t u

f t f u	

u t u u	

Consequently, the truth tables for $\&$ and $=>$ are:

$\&$ t f u	$=>$ t f u
-----	-----
t t f u	t t f u
-----	-----
f f f u	f t t t
-----	-----
u u f u	u t u u

We furthermore say that a universally quantified statement, $\text{Forall } x P(x)$, is true if $P(x)$ is true for all x , false if there exists at least one x for which $P(x)$ is false, and undefined otherwise. Consequently, $\text{Exists } x P(x)$ is true if $P(x)$ is true for at least one x , false if $P(x)$ is false for every x , and undefined otherwise.

Kripke evaluates statements in his logic according to the rules of these truth tables, although his 'third value' is semantically different from Kleene's. He does not assign the third value to statements whose truth values are unknown, but to statements whose truth values *cannot* be decided on, by the very nature of the statement and its relation to the actual world.

In particular, Kripke sets up his system so that ordinary, easily understandable sentences like 'snow is white' and 'it is true that 'grass is green' is false' are grounded and thus are either true or false, while self referential statements like:

(5) (5) is not true

and

(6) (6) is true

are not grounded and get the third indeterminate truth value. Sentences like (3) or (4) may be grounded or ungrounded, depending on the empirical state of the world.

Like Tarski, Kripke considers a hierarchy of languages. However, there is only one truth predicate in all of these languages: $T(x)$. He explains the construction of his hierarchy by considering how we might explain the concept of truth to someone who had never been exposed to the idea before. We might say that we are willing to state that a sentence is true [false] precisely under those circumstances when we are willing to assert [deny] the sentence itself. Now, since the individual has no concept of truth, he would initially be unable to assert or deny the sentence 'snow is white' is true.' But he *can* initially assert the sentence 'snow is white', so upon reflection and using the definition, he can assert 'snow is white' is true.' If he can assert this sentence, he can, upon further reflection, assert "'snow is white' is true' is true' and so on. As he continues through further rounds of reflection, he is able to assert more and more sentences.

Any sentence that eventually gets a truth value, whether true or false, in this process is called grounded. If it never gets a truth value, it is ungrounded. Clearly, a sentence such as

(6) (6) is true

will never get a truth value and is thus ungrounded.

Formally, we proceed as follows: We start with a classical first order language L and extend it to a language L by adding our truth predicate $T(x)$. An interpretation of $T(x)$ is given by the partial set (S_1, S_2) where S_1 is the extension of $T(x)$, S_2 is the antiextension of $T(x)$, and $T(x)$ is undefined for entities outside $S_1 \cup S_2$. We let $L(S_1, S_2)$ be the interpretation of L which results from interpreting $T(x)$ by (S_1, S_2) .

The first language in our hierarchy, L_0 , is defined as $L(\phi, \phi)$. That is, L_0 is the language where $T(x)$ is completely undefined. $L_1 = L(S_1, S_2)$, where S_1 is the set of codes (i.e. Gödel numbers) of true sentences of L_0 and S_2 is the set of all elements of our domain D which either are not codes of sentences of L_0 or are codes of false sentences of L_0 . This process continues as we construct our hierarchy.

In general, for each ordinal α , L_α is defined as follows:

If α is a successor ordinal (i.e. there exists a β such that $\alpha = \beta + 1$), then

$L_\alpha = L(S_{1,\alpha}, S_{2,\alpha})$, where $S_{1,\alpha}$ is the set of codes of true sentences of L_β and $S_{2,\alpha}$ is the set consisting of all elements of D which either are codes of false sentences of L_β or are not codes of sentences of L_β .

If α is a limit ordinal, then

$L_\alpha = L(S_{1,\alpha}, S_{2,\alpha})$ where

$$S_{1,\alpha} = \bigcup_{\beta < \alpha} S_{1,\beta} \text{ and}$$

$$S_{2,\alpha} = \bigcup_{\beta < \alpha} S_{2,\beta}.$$

At successor levels, we take the truth predicates over the previous level; at limit levels we take the union of all sentences declared true or false at previous levels.

There are two important things to note about this construction. The first is that as α increases, the extension and antiextension of $T(x)$ increase. So, once a sentence gets a truth value at some level during the construction, it keeps that truth value. The second is that this increasing is not strict. We will eventually get to a level where no new sentences are being decided. This means that there will be a level σ such that $(S_{1,\sigma}, S_{2,\sigma}) = (S_{1,\sigma-1}, S_{2,\sigma-1})$. We will call such a σ a *fixed point* of L . The proof that such a fixed point exists is straightforward. All the sentences of L form a set, which must have some cardinality c ; that

is, the number of sentences in L is c . If there were no fixed point, we would have the sequence

$$L_0 \subset L_1 \subset \dots L_\omega \subset \dots$$

with each inclusion being strict. We would thus eventually reach a point at which we would have some L_α with more than c sentences, which is a contradiction. Thus a fixed point must exist.

Once we get to this level, we will not be able to assign truth values to any more sentences, no matter how far we continue the construction. We now call any sentence that has been assigned a truth value in this process 'grounded'; all other sentences are 'ungrounded.'

Kripke has thus shown how to construct a language L_σ which contains its own truth predicate and which at the same time avoids paradox.

3.4. Relevance of Kripke's Work to a Logic of Knowledge

Kripke's approach to truth is especially interesting to us at this point because it suggests ways of avoiding the Montague-Kaplan paradox. Our strategy for avoiding such paradoxes will be to construct a language in which sentences like

$$(1) S \text{ iff Know}(a, 'S')$$

(along with of course such sentences as (5) and (6)) are not grounded.

Like Kripke, we will construct a hierarchy of languages, building up towards a language which contains its own truth and knowledge predicates. We will be explaining knowledge as true belief, though for the purposes of the construction we could as well have taken knowledge to be true justified belief. We adopt the former strategy purely for reasons of simplicity. The only point crucial to this construction is that knowledge not be taken as a primitive concept.

We will start off with a classical first order language L , which comes with a fixed set of predicates and relations, including the relation believe(a, p) which means that agent a believes the proposition denoted by p . Believe(a, p) is a well-defined function on all numbers p . [3]

Our first language L_0 does not contain the concepts of truth or knowledge so we can formulate statements such as 'snow is white' or 'Berlioz is dead,' but not "'snow is white' is true" or 'John knows 'Berlioz is dead.'" Of course, as we ascend the hierarchy, these statements will get a truth value. In L_1 we will be able to say "'snow is white' is true" or 'John knows 'snow is white"'; in L_2 we will be able to say 'John knows "'snow is white' is true"' and so on.

In general, we can say that for each (successor ordinal) i ,

- Know(a, p) has a positive truth value in L_i , i.e. a knows that p , if

- (i) p has a positive truth value in L_{i-1} and
- (ii) a believes p

- Know(a, p) has a negative truth value in L_i if

- (i) p has a negative truth value in L_{i-1} or
- (ii) p has a positive truth value in L_{i-1} but a does not believe p

- Know(a, p) has an undefined truth value otherwise. (In particular, Know(a, p) will be undefined in L_i if p is undefined in L_{i-1} .)

3- To avoid the Believer Paradox that such a move entails [Thomason 1980], we reject the 'assumption of arrogance': Believe($a, \text{'Believe}(a, p) \Rightarrow p$ ').

Formally, we proceed as follows:

We start with our classical first order language L and extend it to a language L by introducing the truth predicate $T(x)$ and the knowledge predicate $\text{Know}(a,x)$. The interpretation of $T(x)$ is given as before; the interpretation of $\text{Know}(a,x)$ is given by the partial set (K_1, K_2) where K_1 is the extension of $\bigcup_a \text{Know}(a,x) = \{x \mid \text{Exists a } \text{Know}(a,x)\}$ - i.e., K is the set of all known sentences, K_2 is its antiextension, and for all a , $\text{Know}(a,x)$ is undefined for entities outside $K_1 \cup K_2$. $L(S_1, S_2, K_1, K_2)$ is the interpretation of L which results from interpreting $T(x)$ by (S_1, S_2) and $\text{Know}(a,x)$ by (K_1, K_2) . Of course, $L_0 = L(\phi, \phi, \phi, \phi)$ since both $T(x)$ and $\text{Know}(a,x)$ are completely undefined at this point.

Thus for each ordinal α ,

if α is a successor ordinal (there exists a β such that $\alpha = \beta + 1$) then

$$L_\alpha = L(S_{1,\alpha}, S_{2,\alpha}, \bigcup_a K_{1,a,\alpha}, \bigcup_a K_{2,a,\alpha}) \text{ where}$$

$S_{1,\alpha}$ and $S_{2,\alpha}$ are defined as before, and for each a , $K_{1,a,\alpha}$ is the set of codes of true sentences θ where θ is true in L_β and $\text{believe}(a,\theta)$ is true in L_α and, for each a , $K_{2,a,\alpha}$ is the set of all elements θ in D where

- [1] θ is the code of a false sentence of L_β or θ is the code of a true sentence of L_β but $\text{believe}(a,\theta)$ is false in L_β or
- [2] θ is not the code of a sentence of L_β .

If α is a limit ordinal, let

$$L_\alpha = L(S_{1,\alpha}, S_{2,\alpha}, \bigcup_a K_{1,a,\alpha}, \bigcup_a K_{2,a,\alpha}),$$

where as before, $S_{1,\alpha} = \bigcup_{\beta < \alpha} S_{1,\beta}$

$S_{2,\alpha} = \bigcup_{\beta < \alpha} S_{2,\beta}$ and where for any a ,

$$K_{1,a,\alpha} = \bigcup_{\beta < \alpha} K_{1,a,\beta} \text{ and}$$

$$K_{2,a,\alpha} = \bigcup_{\beta < \alpha} K_{2,a,\beta}.$$

As before, we can show that the extension of $T(x)$ and $\bigcup_a \text{Know}(a,x)$ in L_α is non-decreasing as α increases, so that as we proceed, more and more statements are assigned truth values. Moreover, we can prove that there exists a fixed point σ such that the extension of $T(x)$ and $\bigcup_a \text{Know}(a,x)$ remain the same for L_σ and $L_{\sigma-1}$. We call a sentence grounded if it has been assigned a truth value at L_σ ; otherwise it is ungrounded.

It is easy to see that problematic sentences like

$$(1) S \text{ iff } \text{Know}(a, 'S')$$

as well as sentences such as

$$(7) S \text{ iff } \text{Know}(a, 'S')$$

will be ungrounded in such a construction, and that the Knower Paradox is therefore resolved. Thus we have successfully avoided the Kaplan-Montague paradox and have dealt with the primary objection to a first order logic that treats 'know' as a syntactic predicate.

3.5. Some Unusual Results

It should be noted at this point that although we have demonstrated that we can construct a consistent first order language that contains 'Know' as a syntactic predicate, our resulting logic is not a classical logic. This is true of Kripke's system as well as ours and is a direct consequence of using a three valued logic such as Kleene's. This is because we do not have the law of the excluded middle: since a statement can be true, false, or of indeterminate truth value, we do not have the axiom schema $p \vee \neg p$. It is simply not true that every statement is true or false; ungrounded statements are neither.

Losing the law of the excluded middle has far reaching consequences. For starters, we no longer have the rules $p \Rightarrow p$ or $\neg(p \ \& \ \neg p)$ since these are equivalent to $p \vee \neg p$. Likewise we do not have the quotation stripper axiom schema $\text{True}('p') \Rightarrow p$ or the adder axiom $p \Rightarrow \text{True}('p')$. If p happens to be ungrounded, it is not the case that $\neg \text{True}('p') \vee p$ is true or that $\neg p \vee \text{True}('p')$ is true. Furthermore, it is not the case in our system that $\text{Know}(a, 'p') \Rightarrow p$ for all p .

The situation is not as bleak as it might initially appear, however. All the classical laws of logic do hold as long as we are dealing with grounded statements, they only fail for ungrounded statements. Since in practice we will be dealing only with grounded statements, the failure of the classical laws of logic need not concern us. Moreover, Kripke's system and our system both have the classical *inference* rules: $\text{True}('p')/p$ and $p/\text{True}('p')$. Our system also has the inference rule $\text{Know}(a, 'p')/p$.

Nonetheless, the loss of the laws of classical logic entailed by a move to a three valued logic have prompted a number of researchers to reject Kripke's solution to the Liar Paradox. Don Perlis, Anil Gupta, and Hans Herzberger have independently attempted to resolve the Liar Paradox within classical two valued first order logic [Perlis 1981, Gupta 1982, Herzberger 1982]. While the law of the excluded middle does hold in their systems, they all lose the 'meta' law of the excluded middle: $\text{True}('p') \vee \neg \text{True}('p')$. They likewise lose the classical inference rule $p/\text{True}('p')$. Whether these systems can thus be characterized as classical is debatable. They are nevertheless of theoretical interest, and in particular may lend themselves to other resolutions to the Knower Paradox. [See Asher and Kamp, 1986].

4. A First Order Theory of Knowledge and Action

Now that we've demonstrated that a syntactic treatment of knowledge can in fact be consistent, we can proceed to develop our first order logic of knowledge and action. Our ultimate goal is a comprehensive theory of knowledge, action, and communication, but in this section we will limit ourselves to the types of problems that Moore considered in his system. In particular, we will show how we can integrate Moore's solution to the problem of knowledge preconditions into our first order theory, and demonstrate how agents can reason about their ability to do an action.

4.1 Some Preliminary notation

We will be working with a classical first order language with all the usual logical connectives such as And, Or, Not, $\Rightarrow$. Variables are in lower case; constants begin with upper case or are numbers. Our ontology for actions will be loosely based on the temporal logic developed by Drew McDermott [McDermott 1982, 1985], although the particular ontology chosen is not crucial to the limited theory presented here. [4] We will regard time as an infinite line branching into the future. Like the real number line, this time line contains an infinite number of points. Each point on the line corresponds to a state or situation in the world- i.e., a 'snapshot' of the world at a given instant in time. An interval is specified by two points and refers to the time interval spanned by two instants of time.

An action is defined as a collection of intervals, roughly speaking, those in which the action takes place. For example, 'play-baseball' is the set of all intervals of minimum length in which baseball is played. 'Do' is a function which maps an agent and an action into an *event*, the collection of intervals in which an agent does a particular action. For example, $\text{Do}(\text{Susan}, \text{play-baseball})$ is the set of all intervals in which Susan does the action of playing baseball. If we want to speak about a particular instance of an action or event, we specify the actual interval in which it takes place, e.g., $\text{Do}(\text{Susan}, \text{play-baseball}, [S10, S36])$. R is a

4- The ontology chosen will, however, be of critical importance to the development of the theory of planning and action presented in [Morgenstern 1987].

predicate which takes an event and two situations, and says that the result of the event's occurring in the first situation is the second situation. Thus, $R(\text{Do}(\text{John}, \text{feed}(\text{Felix-cat})), S2, S4)$ is true exactly when $S4$ is the situation that results from John feeding Felix the Cat in $S2$.

In general, every predicate must be qualified by a situation or interval. $\text{Standing}(\text{John}, S5)$ means that John is standing in situation $S5$, while $\text{standing}(\text{John}, [S3, S22])$ means that he is standing during the entire interval $[S3, S22]$. In practice, if a predicate is true over all situations, we may drop the situational argument, so that we can write, for example, $\text{white}(\text{snow})$ instead of $\text{white}(\text{snow}, S11)$ or $\text{white}(\text{snow}, [S11, S15])$. This is just shorthand, however; white is still a two-place predicate. In general, we will use $T('P(x_1, \dots, x_n)', s)$ and $P(x_1, \dots, x_n, s)$ interchangeably.

4.1.2 Dealing with Strings

Any expression x of L can be mapped into a string, or number, that represents x using the Godelization function G . G can be any function that maps unique expressions of L into unique numbers. In general, we write $G(x)$ as ' x '. G is invertible; given any string, we can recover the formula it represents.

Assuming a model M , we can now give our semantics for True and Know :

- (1) $M \models T(p)$ iff $M \models G^{-1}(p)$
- (2) $M \models \text{Know}(a, p, s)$ iff $M \models T(p)$ and $M \models \text{Believe}(a, p, s)$

The quotation construct is one of the most important features of L . Strings will serve not only as arguments to attitude predicates such as Know , Believe , and Intend , but also as arguments to communicative acts such as Request and Utter . In addition, we will use strings to obtain the expressivity of a higher order language, while keeping L first order. Back in Section 2, we saw that we could get the effect of quantifying over sentences by quantifying over the strings that represented these sentences. Thus we could say that John knew something that Bill didn't know by saying:

Exists x $\text{Know}(\text{John}, x, s)$ and Not $\text{Know}(\text{Bill}, x, s)$

Similarly, we can effectively quantify over predicates and functions by quantifying over the strings that represent them. Since strings are just numbers, we may quantify over them in a first order language.

String manipulation, however, turns out to be a non-trivial problem. We can easily formulate such simple sentences as

$\text{Know}(\text{John}, \text{'Lives}(\text{brother}(\text{Mary}), \text{Cleveland}, S4)', S4)$

More complicated constructs, however, involving quantification into quoted contexts are problematical.

Suppose we wish to say, for example, that people always know they're alive. Since we want to make a universal statement, we know that we will have to use universal quantifiers. So, a naive first attempt might be:

forall a, s ($\text{Alive}(a, s) \Rightarrow \text{Know}(a, \text{'alive}(a, s)', s)$)

This is clearly wrong, however; if we substitute John for a and $S1$ for s , we get

$\text{Alive}(\text{John}, S1) \Rightarrow \text{Know}(\text{John}, \text{'Alive}(a, s)', S1)$

which, of course, is nonsense and not at all what we want. We want to get the *effect* of

```

Know(John,'Alive(John,S1)',S1)
Know(John,'Alive(John,S33)',S33)
Know(Mary,'Alive(Mary,S33)',S33)

```

and so on, but of course we wish to state the principle as *one* universal statement.

The problem, of course, is that we can't quantify into quoted contexts; the quotes are in some sense opaque. Still, in order to achieve full expressivity for L , we will need the ability to pull apart and put together strings in arbitrary manners. The three functions of L introduced below help us achieve that purpose. The first and third map strings into strings; the second maps terms into strings:

(1) *concat*. *concat* denotes the concatenation function of L : it takes any number of strings as its arguments and returns as its value a single string, formed by joining all the argument strings together. So, for example,

```

concat('Hef','fal','ump') = 'Heffalump'
concat('At(X,Y,', 'S)') = 'At(X,Y,S)'

```

(2) *name-of*. The *name-of* function maps a term into a string which uniquely names that object. Although the *name-of* function is a bit of an idealization, we justify its existence in L by noting that in everyday usage, for every term of L referring to an actual object, there usually exists exactly one constant which designates that term. [5] For example, the constant Ronald Reagan uniquely designates the term *president(U.S., 1986)* [6], the constant Sarah Ferguson uniquely designates the term *bride(Prince-Andrew,7-23-86)*. The *name-of* function returns the Godel function of the constant that designates the term. Thus:

```

name-of(president(U.S.,1986)) = 'Ronald Reagan'
name-of(bride(Prince Andrew,7-23-1986)) = 'Sarah Ferguson'
name-of(brother(Mary)) = 'Bill'
name-of(John) = 'John'

```

Note that the *name-of* function applied to a constant always yields the quoted constant.

(3) *g*. The *g* function takes a string as its argument and returns that string, quoted, as its result. For example,

```

g('John') = "John"
g('brother(Mary)') = "brother(Mary)"

```

g can be thought of as a partial reflection of G into L . *g* is of course not as general as G ; it is defined only on strings.

We use *g* when we wish to talk about the construct of quotation with L . Its usefulness will be shown below.

These three functions combine to give L full expressivity. For example, we can express the fact that all agents know that they are alive in the following manner:

```
forall a,s (Know(a, concat('Alive(',name-of(a),',',name-of(s),')'),s))

```

If we substitute John for *a* and S1 for *s*, this now works out correctly:

5- Of course, this is not always the case. The classic counterexample is that of the planet Venus, which has two constants designating it: Hesperus and Phosphorus. If we wish L to be able to handle such cases, we could make *name-of* a multi-valued function. In that case *name-of* would evaluate to *one* of the strings naming the object. To simplify the presentation here, however, we make *name-of* a genuine single valued function

6- We ignore the obvious problem presented by such terms as *President(U.S.,1980)* or even *President(U.S.,1-21-1980)* by noting that the temporal argument used here is simply too coarse-grained.

Know(John, concat('Alive(', 'John', ',', 'S1', ')'), S1)

which is

Know(John, 'Alive(John, S1)', S1)

which is precisely what we want.

For another, slightly more complicated example, we express the principle of positive introspection: if an agent knows something, he knows that he knows it. In L , this becomes:

forall a,p,s (Know(a,p,s) =>
Know(a,concat('Know(', name-of(a), ',', g(p), ',', name-of(s), ')'), s)

As should be evident by now, saying reasonably simple things in L quickly becomes unreasonably complicated. The concat notation is fundamentally clumsy; it is difficult to read, write, and understand. We need some way of capturing in L the ease of indirect quotation which occurs with such frequency in natural language. We accomplish this by introducing three abbreviators which are defined in terms of concat, name-of, and g. These abbreviators are all anti-quote operators: their occurrence within a quoted string is a signal to take as unquoted the terms following the operator. These operators are discussed below:

(a) the $\hat{\hat{}}$ abbreviator. The $\hat{\hat{}}$ abbreviator is used when we wish to quantify *over* strings *into* a quoted string. For example, suppose we wish to say that the last letter of the last name of the current president is 'n'. The correct but cumbersome way to express this is:

exists p (last-name(president(U.S., 1986)) = concat(p, 'n'))

In order to do away with the clumsy concat construction, we introduce the antiquote abbreviator $\hat{}$, where for all p, the occurrence of $\hat{p}$ within quotation marks tells us to substitute the expression that p represents inside the string. That is, any string of the form ' $\hat{a}\hat{p}\hat{b}$ ' should be understood to mean concat('a', p, 'b').

If, for example, p is 'xyz', ' $\hat{a}\hat{p}\hat{b}$ ' is 'axyzb'; if p is 'q', ' $\hat{a}\hat{p}\hat{b}$ ' is 'aqb'.

We may now rewrite our axiom as

exists p (last-name(president(U.S., 1986)) = ' $\hat{p}\hat{n}$ '))

Note that $\hat{}$ is not a symbol of L . Thus ' $\hat{p}\hat{n}$ ' is not an expression of L , and $G(\hat{p}\hat{n})$ is meaningless. The $\hat{}$ functions simply as an abbreviation.

The $\hat{}$ abbreviation symbol appears in pairs. In general we antiquote what we find between the odd numbered and even numbered occurrences of $\hat{}$ and quote what we find between the even numbered and odd numbered occurrences of $\hat{}$.

Thus, in exists p,q,r (r = ' $\hat{p}\hat{q}\hat{p}\hat{q}\hat{q}\hat{p}\hat{q}$ '), the first and third occurrences of p are quoted, the second is antiquoted; the first and second occurrences of q are quoted, the third and fourth are antiquoted.

(b) the $!!$ abbreviator. Like the $\hat{\hat{}}$ abbreviator, the $!!$ abbreviator is used when we wish to quantify into strings. However, it is interpreted differently. If p is 'xyz', ' $\hat{a}\hat{p}\hat{b}$ ' is 'axyzb', while ' $\hat{a}!p!b$ ' is ' $\hat{a}xyzb$ '. That is, when we see $\hat{p}$, we substitute the expression that p represents, when we see $!p!$ we substitute p itself. Any string of the form ' $\hat{a}!p!b$ ' can be defined as: concat('a', g(p), 'b'). In fact, we can define $!p!$ as ' $\hat{g(p)}$ '.

The $!!$ abbreviator is useful when we wish to refer to entire propositions within a quoted context. For example, suppose we wish to state the principle of positive introspection, applied to John in situation S1: If John knows p, then he knows that he knows p.

We write

forall p (Know(John, p, S1) => Know(John, 'Know(John, $!p!$, S1)', S1))

If p is 'White(Snow)' this expands to

$\text{Know}(\text{John}, \text{'White(Snow)'}, S1) = > \text{Know}(\text{John}, \text{'Know}(\text{John}, \text{'White(Snow)'}, S1)', S1)$

which is what we want.

Had we used the $\wedge$ instead of the $!$, we would have obtained:

$\text{Know}(\text{John}, \text{'White(Snow)'}, S1) = > \text{Know}(\text{John}, \text{'Know}(\text{John}, \text{White(Snow)}, S1)', S1),$

which is ill-formed and thus meaningless.

(3) the @ abbreviator. The @ abbreviator is used when we wish to refer to the standard name of an object within a string. When applied to an object (constant or term) it returns the standard name of the object, antiquoted. Thus ' $a@xb$ ' is to be understood as $\text{concat}('a', \text{name-of}(x), 'b')$.

Thus, the general principle of positive introspection, that if an agent knows p , he knows that he knows p , can be expressed as:

forall a, p, s
 $\text{Know}(a, p, s) = > \text{Know}(a, \text{'Know}(@a, !p!, @s)', s)$

If a is John, p is 'White(Snow)' and s is $S1$, this expands to

$\text{Know}(\text{John}, \text{'White(Snow)'}, S1) = >$
 $\text{Know}(\text{John}, \text{'Know('name-of(John)', !p!, 'name-of(S1)'')', S1)$

which expands to

$\text{Know}(\text{John}, \text{'White(Snow)'}, S1) = >$
 $\text{Know}(\text{John}, \text{'Know}(\text{John}, \text{'White(Snow)'}, S1), S1).$

4.2 The Actual Theory

4.2.1 Axioms on Knowledge

We can now present some axiom schemata that capture our basic intuitions on knowledge. Not surprisingly, these schemata are rather similar to Moore's M1-M5; whether we treat 'know' as a modal operator or as a syntactic predicate, our basic intuitions on knowledge remain the same. It should be noted that the schemata below, particularly K2-K4, hold only for grounded statements.

K1: Axioms of ordinary predicate logic (see [Mendelson 1964], Section 2.3)

K2: $\text{Know}(a, p, s) = > p$ (veridicality)

K3: $\text{Know}(a, p, s) = > \text{Know}(a, \text{'Know}(@a, !p!, @s)', s)$ (positive introspection)

K4: Agents know the rules of inference:

(i) $\text{Know}(a, !p! = > !q!', s)$ and $\text{Know}(a, p, s) = > \text{Know}(a, q, s)$
 (Modus Ponens)

(ii) $\text{Know}(a, 'p', s) = > \text{Know}(a, \text{'Forall } x (p)', s)$
 (Generalization)

K5a: If p is an axiom of predicate logic then $\text{Know}(a, p, s)$ for any s

K5b: $\text{Know}(a, K1-K5b, s)$ for any s

It will be noticed that K1-K5 differ from M1-M5 in two important ways:

(1) K1 and K4 are quite a bit stronger than their counterparts, M1-M4. M1 and M4 just specify that all axioms of propositional logic are in the system and that agents can and do use the rule of inference Modus Ponens. Thus, the system supports only inferences made via the rules of propositional logic, and assumes that agents are limited to these inferences as well. In contrast, K1 specifies that all axioms of predicate logic are in the system, and K4 posits that agents can and do reason with the rules of inference of predicate logic. [7] Our system is

7- Of course, the axioms and inference rules are not fixed; we may decide to use a different set, such as is presented in [Kleene 1955], or choose a natural deduction system, as in [Mates 1972], where there

thus considerably more powerful. Traditionally, modal logic has restricted itself to the propositional calculus since the introduction of quantifiers poses so many problems ([Marcus 1971], [Kaplan 1971], [Hughes and Cresswell 1968]). Of course, not much can be said without predicate calculus; Moore's system, however, lets in predicate calculus through the back door since the actual proofs of theorems are carried out in the first order theory of possible worlds. We, on the other hand, have to explicitly assert all the axioms of predicate calculus and assume that all agents can and do reason with these principles. These axioms are strong enough to prove that all agents know all logical consequences of their knowledge, and that they thus behave like 'perfect reasoners.'

(2) On the other hand, K5 is a good deal weaker than M5. As we have said previously, we do not wish to assume that all agents know all axioms, since there are many axioms on the world that agents simply aren't aware of. We do, however, wish to assume that all agents know all axioms of logic and the basic axioms of knowledge. K5a and K5b capture these constraints, thus avoiding the overly strong assumptions on knowledge that modal logics and possible worlds semantics entail.

4.2.2 Solution to the Knowledge Preconditions Problem

We are now ready to tackle the central problem of our theory: When can we say that an agent knows how to do an action?

Moore proposed that an agent knows how to do an action if he knows an executable description for the action, or by implication, if he knows a rigid designator for the action. We argued in section 2 that while the first condition is reasonable, the second seems implausible. There is a certain sense, nonetheless, in which Moore's proposal does capture some of our intuitions about knowledge and action. Moore, it will be remembered, believed that most simple action-types are rigid functions, or basic procedures that any individual knows how to perform. So an individual can do a simple action if he knows rigid designators for the parameters of the action. Furthermore, complex actions-types are built up out of primitive action-types in ways that are known to every agent, so that knowing how to do a complex action also reduces to knowing rigid designators for parameters of basic action-types.

It seems clear that there is some truth to this theory. It is true that all agents know how to perform certain primitive action-types such as lift or carry, and that they know how to do the particular action lift(x) if they know some fixed object or 'rigid designator' for x. It is not true, however, that every agent knows how complex action-types are built up out of primitive action-types. For example, we can think of the complex action-type sharpen(pencil,sharpener) as being defined as:

sequence(place-in(pencil,sharpener), rotate(pencil))

where place-in and rotate are primitive action-types. But we cannot assume that every agent knows this procedure; in particular, children don't learn it until they're three or four, and people from distant civilizations may be completely unfamiliar with the procedure. We could, however, say that an agent knows how to sharpen a pencil if he *knows* the basic procedure and moreover knows fixed objects- or constants- for the pencil and sharpener.

We are saying, then, that an agent knows how to do an action if

- (a) the action type is primitive and he knows constants for the parameters of the action
- or if
- (b) he knows how the action is built up out of simpler actions and he knows how to perform those simpler actions.

are no logical axioms, but more rules of inference. For ease of presentation, we have chosen a system with the minimal number of inference rules.

We set down our axioms for Know-how-to-perform below. Most of the primitives which appear in these axioms, such as $\text{sequence}(a_1, \dots, a_n)$ or $\text{while}(p, \text{act})$ are self explanatory. The predicate $\text{stidstr}()$ ranges over strings and is true of a string if and only if the string is the quoted form of a standard identifier of L . In general, $\text{stidstr}(p)$ iff exists x ($\text{name-of}(x) = p$). Thus, $\text{stidstr}(\text{'Mary'})$ and $\text{stidstr}(\text{'123'})$ are true; $\text{stidstr}(\text{'brother(Mary)'})$ and $\text{stidstr}(\text{'telephone-number(Mary)'})$ are false.

KP1:

$\text{k-h-t-p}(a, '@ f(@ \text{arg1}, \dots, @ \text{argn})', s)$
 if
 primitive-action(f) and
 exists p_1 ($\text{stidstr}(p_1)$ and $\text{Know}(a, \text{'equal'('p1', @ \text{arg1}, @ s)', s})$
 and and
 exists pn ($\text{stidstr}(pn)$ and $\text{Know}(a, \text{'equal'('pn', @ \text{argn}, @ s)', s})$

KP2:

$\text{k-h-t-p}(a, '@ f(@ \text{arg1}, \dots, @ \text{argn})')$
 if
 exists $f^*, \text{arg1}^*, \dots, \text{argm}^*$
 $\text{Know}(a, '(((@ f(@ \text{arg1}, \dots, @ \text{argn}) = @ f^*(@ \text{arg1}^*, \dots, @ \text{argm}^*)), s)$
 and $\text{k-h-t-p}(a, '@ f^*(@ \text{arg1}^*, \dots, @ \text{argm}^*'))$

KP3:

$\text{k-h-t-p}(a, \text{'seq'('act1', 'act2')}, s)$
 if
 $\text{k-h-t-p}(a, \text{act1}, s)$ and
 $R(\text{Do}(a, \text{act1}), s, s^*) = >$
 $\text{k-h-t-p}(a, \text{act2}, s^*)$

KP4:

$\text{k-h-t-p}(a, \text{'cond}(p, \text{'act1'}, \text{'act2'}), s)$
 if
 $\text{Know}(a, \text{'p'}, s)$ and $\text{k-h-t-p}(a, \text{act1}, s)$
 or $\text{Know}(a, \text{'p'}, s)$ and $\text{k-h-t-p}(a, \text{act2}, s)$

KP5:

$\text{k-h-t-p}(a, \text{'while}(p, \text{'act'})', s)$
 if
 $\text{k-h-t-p}(a, \text{'cond}(p, \text{'seq'('act', while}(p, \text{'act'}, \text{nil}))', s)$

Note that in our system, as in Moore's, an agent need not be able to perform an entire action sequence when he starts out doing an action. Instead, he can pick up information as he goes along. So to do the action sequence ($\text{Look up Bill's phone number}, X; \text{Dial}(X)$), it is not necessary that John be able to perform $\text{Dial}(X)$. In fact, he will not be able to do this action when he starts out, because he won't know a constant, or number, for X . He will, however, be able to perform $\text{Dial}(X)$ as soon as he completes $\text{Look-up}(\text{Bill's phone number})$ and this is all that we require. Agents must be able to execute each step in an action sequence as they get to that step.

We also define the predicate Know-how-to-achieve, which enables an agent to reason about accomplishing a particular goal.

Informally, an agent knows how to achieve a situation with a certain property, if he knows

For

Know how

Know(e 'Forest') : 2 (P(d₀(G) : 1)

and

De-

Know-how-to-perform(2 act 61)

None of this suggests that developing a theory of communication will be an easy or trivial task. There are many issues that will have to be worked out before we can tackle even an easy problem such as our thumbs-in-the-kitchen John's plan to learn how to bake. We will have to formalize the relationship between knowledge and communication, indicating how knowledge constrains the contents of communications, and how communication can affect an agent's knowledge. We will have to describe the interrelationships among friendly agents and the protocols that these agents obey. For more complex plans, we will have to allow hierarchical environments in which an agent may dominate a number of subordinate agents, who may be more knowledgeable than he is. As we continue to consider the types of plans that our system should handle, the requirements for our theory will grow, and our projected theory becomes more complex.

Nonetheless, our observations do suggest that our current theory of knowledge and action can be extended to a considerably complex theory of communication. We have thus far constructed only the core of a larger theory, but it is flexible and robust enough to allow for development and expansion in many directions.

Acknowledgements

This research was supported in part by NSF grant DCR-8603758. I'd like to thank Ernie Davis for ideas, suggestions, and guidance, and Jerry Hobbs, David Israel, Kurt Konolige, Bob Moore, and Moshe Vardi for helpful discussions and criticisms.

Appendix

In his 1980 thesis, Moore demonstrated that his logical formalism could effectively handle three benchmark problems concerning the interrelated effects of knowledge and action. Here, we show that our system is capable of handling those same problems in an efficient manner, and thus that in some sense our system subsumes his.

First, we present a short summary of the three benchmark problems. The interested reader is referred to [Moore 80] for more details.

- (1) We wish to show that an agent can do an action if he knows fixed objects (rigid designators in Moore's system; standard identifiers in ours) for the parameters of the action, and that he can achieve a goal if he knows of some action to achieve that goal and can do the action. In particular, we show that if John knows standard identifiers for a particular safe and for the combination of that safe, that he can dial the combination of the safe, thus opening the safe.
- (2) We wish to formalize the notion of a test. Generally, a test is an action that results in our knowing whether a particular condition is true or not. For example, if we do not know whether a particular solution is acid or base, we can test it by performing the action of inserting litmus paper into the solution. If we know how the acidity of a solution affects the color of litmus paper, we will know after we perform the action whether or not the solution is acidic. If the action results in the litmus paper turning pink, the solution is acid; if the litmus paper turns blue, the solution is base.

Here, we show that John can determine whether a particular number is the combination of a safe by performing the 'test' of dialing the combination on the initially closed safe. If the safe opens, then John knows that he did in fact dial the combination; if it remains closed, he knows that the number he dialed was not the combination.

- (3) Finally, we show that John can perform a sequence of actions, even if initially he cannot perform every action in the sequence. We show that even if John does not know the combination of a safe when he starts out, he can perform the sequence of Read(paper);Dial(combination), if the combination of the safe is written on the piece of paper.

The basic kernel of our theory- the axioms on Know-how-to-perform and Know-how-to-achieve- have been presented in the paper. Below, we introduce a number of useful general axioms, and then proceed to give the domain axioms on dialing, reading, and the like.

E1: Forall x, s
Equal(x, x, s)

E2: Forall a, s
Know($a, E1, s$)

E1 states that every object is always equal to itself; E2 states that all agents know E1.

A1: Forall a, loc, s
 $At(a, loc, s) \Rightarrow Know(a, 'At(@ a, @ loc, @ s)', s)$

A1 is a useful simplification which states that agents always know where they are.

I1: Forall a,act,s1,s2
 $\text{Intend}(a, \text{Do}(a, \text{act}), s1) \text{ and } R(\text{Do}(a, \text{act}), s1, s2) \Rightarrow$
 $\text{Know}(a, 'R(\text{do}(@ a, @ \text{act}), @ s1, @ s2)', s2)$

I1 says that if an agent intends to do an action and does it, then he knows in the resulting situation that he has done the action.

We now present some axioms on Dial. Dial is a two place function taking a combination to be dialed and a safe upon which the combination is dialed. These are just Moore's axioms recast into our formalism.

D1: Forall a,x1,x2,s
 $\text{Precondsat}(\text{Do}(a, \text{Dial}(x1, x2)), s) \Leftrightarrow \text{At}(a, x2, s)$

D2: Forall a,x1,x2,s1,s2
 $R(\text{Do}(a, \text{Dial}(x1, x2)), s1, s2) \Rightarrow$
 $\text{Equal}(x1, \text{comb}(x2), s1) \Rightarrow \text{Open}(x2, s2) \text{ and}$
 $\text{Not Equal}(x1, \text{comb}(x2), s1) \text{ and } \text{Not Open}(x2, s1) \Rightarrow \text{Not Open}(x2, s2) \text{ and}$
 $\text{Open}(x2, s1) \Rightarrow \text{Open}(x2, s2)$

D3: Forall a,x1,x2,s1,s2
 $R(\text{Do}(a, \text{Dial}(x1, x2)), s1, s2) \Rightarrow$
 $\text{Open}(x2, s2) \Rightarrow \text{Know}(a, 'Open(@ x2, @ s2)', s2) \text{ and}$
 $\text{Not Open}(x2, s2) \Rightarrow \text{Know}(a, 'Not Open(@ x2, @ s2)', s2)$

D4: Forall a,x1,x2,s1,s2
 $R(\text{Do}(a, (\text{Dial}(x1, x2))), s1, s2) \Rightarrow$
 Forall p
 $p \neq 'Open(@ x2)' \text{ and } p \neq 'Not Open(@ x2)' \Rightarrow$
 $T(p, s1) \Leftrightarrow T(p, s2)$

D5: Primitive-Operation('Dial')

D1 says that the physical precondition for opening a safe is being at the same location as the safe. D2 describes how dialing a number on a safe affects the safe. If the safe is open, it remains open. If the agent dials the combination of the safe, the safe is open as a result. If the safe is initially closed and the number the agent dials is not the combination of the safe, then the safe is closed after the agent finishes dialing. D3 says that after an agent finishes dialing, he knows whether the safe is open or not. D4 is a frame axiom and states that all statements that are true before dialing are true after dialing- with the obvious exceptions, of course, of statements that say whether or not the safe is open. Naturally, these may change in truth value as a result of the dialing action. D5 says that the Dial action is primitive.

Below, we present the proofs. The proofs are done using natural deduction. In addition to the standard rules of inference [see, e.g. Mates 1979], we add the following, justified by Axioms K1-K5:

From $\text{Know}(a, p1 \dots pn, s)$ and $p1 \dots pn \vdash q$, infer $\text{Know}(a, q, s)$.

This is simply the consequential closure principle, and will be denoted as CCP.

We now present the first two proofs:

Proof that John can open a safe if he knows the combination: We assume that John knows the basic axioms on Dial, that he knows some constant for the combination of the safe, and that he is at the safe. We prove that he can open the safe, i.e. that he knows how to achieve the goal of the safe being open. Our proof consists of three parts. First we must show that John

knows-how-to-perform the action of dialing the combination of the safe on the safe. (Steps 1-6). This is easy, since Dial is a primitive operation (Step 1), the particular safe is a constant, and he is assumed to know a constant for the combination of the safe (Step 5). Next, we show that John knows that the physical preconditions for dialing the combination of the safe are satisfied (Steps 7-11). Since John is assumed to be at the safe, he knows that he is at the safe(10), and thus that the physical precondition for this action is fulfilled. Finally, we show that John knows that doing the dialing action will result in the safe being open(12-15), thus completing our proof.

Problem 1:

Given:

Forall s Know(John,D1,s)

Forall s Know(John,D2,s)

Exists x stdstr(x1) and Know(John,'Equal('x',Comb(Sf1),S1',S1)

At(John,Sf1,S1)

Prove:

Know-how-to-achieve(John,'Open(Sf1)',S1)

Proof:

- | | |
|---|-----------|
| 1. Primitive-Operation('Dial') | D5 |
| 2. Exists x1 (stdstr(x1)
and Know(John,'Equal('x1',Comb(Sf1),S1',S1) | Given |
| 3. Equal(Sf1,Sf1,S1) | E1 |
| 4. Know(John,'Equal(Sf1,Sf1,S1)',S1) | 3,E2,CCP |
| 5. Exists x2 (stdstr(x2)
and Know(John,'Equal('x2',Sf1,S1)',S1) | 4 |
| 6. Know-how-to-perform(John,'Dial(Comb(Sf1),Sf1)',S1) | KP1,1,2,4 |
| 7. Know(John,D1,S1) | Given |
| 8. Know(John, 'Precondsat(Do(John,Dial(Comb(Sf1),Sf1),S1) <= >
At(John,Sf1,S1))',S1) | 7,CCP |
| 9. At(John,Sf1,S1) | Given |
| 10. Know(John,'At(John,Sf1,S1)',S1) | 9,A1 |
| 11. Know(John,'Precondsat(Do(John,Dial(Comb(Sf1),Sf1)),S1)',S1) | CCP, 8,10 |

12. Know(John,D2,S1)	Given
13. Know(John,'R(Do(John,Dial(Comb(Sf1),Sf1)),S1,S2) = > Equal(Comb(Sf1),Comb(Sf1),S1) - > Open(Sf1,S2)', S1)	CCP,12
14. Know(John,'Equal(Comb(Sf1),Comb(Sf1),S1)',S1)	CCP,E1,E2
15. Know(John,'R(Do(John,Dial(Comb(Sf1),Sf1)),S1,S2) = > Open(Sf1,S2)',S1)	CCP,13,14
16. Know-how-to-achieve(John, 'Open(Sf1)',S1)	KA1,6,11,15

Proof that if John knows a safe is locked and tries to open it by dialing some number, he will know after he does the dialing whether or not the number he dialed was the combination of the safe.

The proof is divided into two cases, which are proved analogously. In Case I, we assume that the number John is dialing is indeed the combination of the safe. In Case II, we assume that the number John is dialing is not the combination of the safe. In either case, we show that John is aware after dialing whether the number dialed was the combination of the safe.

The proof of Case I proceeds as follows: We assume that John intends to dial a certain number on a safe(Step 1). He thus knows after he has dialed that he has done the dialing(2-3). He also knows that the safe started out closed, and that therefore, if the safe is open in the resulting situation he dialed the correct combination (4-7). He knows that the safe is now open(10). Therefore he knows that the number he dialed was the correct combination when he dialed(11). By the frame axiom, he knows that that number is currently the correct the correct combination (12-13).

Case I:

Given:

Forall s Know(John,D1,s)
 Forall s Know(John,D4,s)
 Know(John,'Not Open(Sf1,S1)',S1)
 Equal(C1,Comb(Sf1),Sf1)
 Intend(John,Do(John,Dial(C1,Sf1)),S1)

Prove:

R(Do(John,Dial(C1,Sf1)),S1,S2) = >
 Know(John, 'Equal(C1,Comb(Sf1),S2)',S2)

Proof:

1. Intend(John, Do(John, Dial(C1, Sf1)), S1)	Given
2. R(Do(John, Dial(C1, Sf1)), S1, S2)	Assumption
3. Know(John, 'R(Do(John, Dial(C1, Sf1)), S1, S2)', S2)	I1, 1, 2
4. Know(John, D2, S2)	Given
5. Know(John, 'Not Open(Sf1, S1)', S1)	Given
6. Know(John, 'Not Open(Sf1, S1)', S2)	Given, D4, D5
7. Know(John, 'R(Do(John, Dial(C1, Sf1)), S1, S2) => Open(Sf1, S2) => Equal(C1, Comb(Sf1), S1)', S2)	CCP, 4
8. Equal(C1, Comb(Sf1), S1)	Given
9. Open(Sf1, S2)	D2, 2, 8
10. Know(John, 'Open(Sf1, S2)', S2)	D3, 2, 9
11. Know(John, 'Equal(C1, Comb(Sf1), S1)', S2)	CCP, 2, 6, 7, 10
12. Know(John, D4, S2)	Given
13. Know(John, 'Equal(C1, Comb(Sf1), S2)', S2)	CCP, 11, 12
14. R(Do(John, Dial(C1, Sf1)), S1, S2) => Know(John, 'Equal(C1, Comb(Sf1), S2)', S2)	Discharge 2, 13

The proof of Case II proceeds along similar lines. We assume that John intends to dial a certain number on a safe(1). He thus knows after he has dialed that he has done the dialing (2-3). He knows that if the safe is not open when he finishes, that the number he dialed was not the combination of the safe (4-5). He knows the safe is now open (8). Therefore, he knows that the number he dialed was not the correct combination of the safe (9). By the frame axiom, he knows that the number is currently not the correct combination of the safe (10-11).

Case II:

Given:

Forall s Know(John,D1,s)

Forall s Know(John,D4,s)

Know(John,'Not Open(Sf1,S1)',S1)

Not Equal(C1,Comb(Sf1),Sf1)

Intend(John,Do(John,Dial(C1,Sf1)),S1)

Prove:

R(Do(John,Dial(C1,Sf1)),S1,S2) =>

Know(John,'Not Equal(C1,Comb(Sf1),S2)',S2)

Proof:

- | | |
|--|----------------|
| 1. Intend(John,Do(John,Dial(C1,Sf1)),S1) | Given |
| 2. R(Do(John,Dial(C1,Sf1)),S1,S2) | Assumption |
| 3. Know(John,'R(Do(John,Dial(C1,Sf1)),S1,S2)',S2) | I1,1,2 |
| 4. Know(John,D2,S2) | Given |
| 5. Know(John,'R(Do(John,Dial(C1,Sf1)),S1,S2) =>
Not Open(Sf1,S2) =>
Not Equal(C1,Comb(Sf1),S1)', S2) | CCP,4 |
| 6. Not Equal(C1,Comb(Sf1),S1) | Given |
| 7. Not Open(Sf1,S2) | D2,2,6 |
| 8. Know(John,'Not Open(Sf1,S2)',S2) | D3,2,7 |
| 9. Know(John,'Not Equal(C1,Comb(Sf1),S1)',S2) | CCP,2,5,8 |
| 10. Know(John,D4,S2) | Given |
| 11. Know(John,'Not Equal(C1,Comb(Sf1),S2)',S2) | CCP,9,10 |
| 12. R(Do(John,Dial(C1,Sf1)),S1,S2) =>
Know(John,'Not Equal(C1,Comb(Sf1),S2)',S2) | Discharge 2,11 |

Proof that John can perform the action sequence of Read(combination); Dial(combination, safe)

We will now show that John can open a safe by reading the combination and dialing that combination, even if he doesn't know the combination of the safe when he begins to do the action. We begin by introducing some axioms on the action Read.

- RD1: Forall a,x,s
Precondsat(Do(a,(Read(x)),s) <=> At(a,x,s)
- RD2: Forall a,x1,x2,s1,s2
R(Do(a,Read(x2)),s1,s2) and Info(x1,x2) =>
Know(a,'Info('x1',@ x2)',s2)
- RD3: Forall a,x1,x2,s1,s2
R(Do(a,Read(x2)),s1,s2) and Info(x1,x2) =>
forall p
p/= 'Know(a,'Info('x1',@ x2)') and p/= 'Not Know(a,'Info('x1',@ x2)') =>
T(p,s1) <=> T(p,s2)
- RD4: Primitive-Operation('Read')

RD1 states the physical precondition for performing the read action. It simply says that an agent must be at the location of the object that he is reading. RD2 says that if some information is written on some object - we assume the information is in the form of a string - and an agent reads that object, then he will know that the information is written on the object. RD3 is the frame axiom for Read and says that the read action changes nothing-except, of course, that an agent will know after the action what information is written on the object that he reads. RD4 states that Read is a primitive action, which is not a true-to-life assumption, but is a useful simplification for our purposes.

We now prove that if there is some piece of paper that has the combination of some safe written on it, and if John knows that the paper has the combination written on it, that John can perform the action sequence of reading and dialing. We first show that John can perform the action of reading the paper (1-5). This is easy since read is a primitive action (1) and he knows a stidstr for the paper (2-4). We then show that reading the paper results in his knowing the information on the paper (6-10), and thus in his knowing the combination of the safe (11-15). Since he knows a stidstr for the safe (16-18) and Dial is a primitive action, he can dial the combination of the safe(20). Thus he can do the entire sequence (21-22).

Given:

Know(John, 'exists c1 (stidstr(c1) and equal(c1,comb(Sf1),S1)',S1)

Prove:

Know-how-to-perform(John,sequence(Read(Ppr1),Dial(Comb(Sf1),Sf1)),S1)

1. Primitive-Operation('Read') R4
2. Equal(Ppr1,Ppr1,S1) E1
3. Know(John,'equal(Ppr1,Ppr1,S1)',S1) E2,CCP,2
4. Exists x1 (stidstr(x1) and
Know(John,'Equal('x1',Ppr1,S1)',S1) 3
5. Know-how-to-perform(John,'Read(Ppr1)',S1) KP1,1,4
6. R(Do(John,Read(Ppr1),S1,S2) Assumption
7. Know(John,'exists c1(stidstr(c1) and equal(c1,Comb(Sf1),S1)
and info(c1,Ppr1)',S1) Given
8. Know(John,'exists c1 (stidstr(c1) and equal(c1,comb(Sf1),S1)
and info(c1,Ppr1)
and info(c1,Ppr1) > equal(c1,comb(Sf1))',S1) CCP,7
9. Exists c1 (stidstr(c1) and equal(c1,comb(Sf1),S1)
and info(c1,Ppr1)) K2,7
- 10.Exists c1 (stidstr(c1) and equal(c1,comb(Sf1),S1) and
info(c1,Ppr1) and know(John,'info('c1',Ppr1)',S2) RD2,6,9
- 11.Exists c1 (stidstr(c1) and equal(c1,comb(Sf1),S1) and info(c1,Ppr1)
and Know(John,'equal('c1',comb(Sf1),S1)',S2)) CCP,8,10
- 12.Forall x (Equal(x,comb(Sf1),S1) => Equal(x,Comb(Sf1),S2)) D4,6
- 13.Know(John,D4) Given
- 14.Know(John,'Forall x (Equal(x,comb(Sf1),S1) =>
Equal(x,Comb(Sf1),S2))',S2) CCP,12,13
- 15.Exists c1 (stidstr(c1) and equal(c1,comb(Sf1),S1) and info(c1,Ppr1)
and Know(John,'equal('c1',comb(Sf1),S2)',S2)) K4,11,14
- 16.Equal(Sf1,Sf1,S2) E1
- 17.Know(John,'Equal(Sf1,Sf1,S2)',S2) E2
- 18.Exists x2 (stidstr(x2) and
Know(John,'equal('x2',Sf1,S2)',S2) 17
- 19.Primitive-Operation(Dial) D5

20. Know-how-to-perform(John,Dial(Comb(Sf1),Sf1),S2)

KP1,15,18,19

21. R(Do(John,Read(Ppr1),S1,S2) =>

Know-how-to-perform(John,'Dial(Comb(Sf1),Sf1)',S2)

Discharge 6,20

22. Know-how-to-perform(John,

sequence(Read(Ppr1,Dial(Comb(Sf1),Sf1)),S1)

KP3,5,21

BIBLIOGRAPHY

- Appelt, Douglas:** *Planning Natural Language Utterances to Satisfy Multiple Goals*, SRI International Technical Note 259, 1982
- Asher, Nicholas and Hans Kamp:** 'The Knower's Paradox and Representational Theories of Attitudes,' *Proc. Conf. on Theoretical Aspects of Reasoning about Knowledge*, Monterey, 1986
- Fikes, R.E. and Nils Nilsson:** 'STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving,' *Artificial Intelligence*, Vol 2, 1971
- Gallin, Daniel:** *Intensional and Higher Order Modal Logics*, American Elsevier, New York, 1975
- Gupta, Anil:** 'Truth and Paradox,' *Journal of Philosophical Logic* Vol. 11, No. 1, 1982
- Herzberger, Hans:** 'Notes on Naive Semantics,' *Journal of Philosophical Logic* Vol. 11, No. 1, 1982
- Hintikka, Jaakko:** *Knowledge and Belief*, Cornell University Press, Ithica 1962
- Hintikka, Jaakko:** 'Semantics for Propositional Attitudes,' in Leonard Linsky, ed. *Reference and Modality*, 1971
- Hobbs, Jerry and Robert Moore:** *Formal Theories of the Commonsense World*, Ablex Publishing Co., Norwood, 1985
- Hughes, G.E. and M.J. Cresswell:** *An Introduction to Modal Logic*, Methuen, London, 1968
- Kaplan, David and Richard Montague:** 'A Paradox Regained,' *Notre Dame Journal of Formal Logic*, vol.1 no.3, pp.79-90, 1960. Also Chapter 9 in **Richmond Thomason**, ed: *Formal Philosophy*
- Kaplan, David:** 'Quantifying In,' in Leonard Linsky, ed. *Reference and Modality*
- Kleene, Stephen C.:** *Introduction to Metamathematics*, Van Nostrand, New York, 1952
- Konolige, Kurt:** 'A First Order Formalization of Knowledge and Action for a Multi-agent Planning System' in J.E.Hays and D.Michie, eds. *Machine Intelligence 10*, 1982
- Kripke, Saul:** *Naming and Necessity*, Harvard University Press, Cambridge, 1972
- Kripke, Saul:** 'Outline of a Theory of Truth,' *Journal of Philosophy*, Vol 72, pp. 690-716, 1975.
- Kripke, Saul:** 'Semantical Analysis of Modal Logic,' *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, Vol. 9, 1963
- Kripke, Saul:** 'Semantical Considerations on Modal Logic,' *Acta Philosophica Fennica*, fasc. 16, pp.83-94, 1963. Also in **Leonard Linsky**, ed: *Reference and Modality*
- Linsky, Leonard**, ed: *Reference and Modality*, Oxford University Press, London 1971
- Marcus, Ruth Barcan:** 'Extensionality' in **Leonard Linsky**, ed. *Reference and Modality*
- Mates, Benson:** *Elementary Logic*, Oxford University Press, 1972
- McCarthy, John and Patrick Hayes:** 'Some Philosophical Problems from the Standpoint of Artificial Intelligence' in Bernard Meltzer, ed: *Machine Intelligence 4*, 1969
- McDermott, Drew:** 'A Temporal Logic for Reasoning About Processes and Plans,' *Cognitive Science*, 1982
- McDermott, Drew:** 'Reasoning About Plans' in Hobbs and Moore, eds. *Formal Theories of the Commonsense World*, 1985
- Mendelson, Eliot:** *Introduction to Mathematical Logic*, Van Nostrand, Princeton, 1964
- Montague, Richard:** 'Syntactical Treatments of Modality with Corollaries on Reflexion Principles and Finite Axiomatizability' in *Acta Philosophica Fennica*, fasc. 16, pp. 153-167 1963. Also in **Richmond Thomason**, ed: *Formal Philosophy*

Moore, Robert: *Reasoning About Knowledge and Action*, SRI Technical Note 191, 1980

Morgenstern, Leora: 'A First Order Theory of Planning, Knowledge, and Action', *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge*, Morgan Kaufmann Publishers, 1986

Morgenstern, Leora: *Foundations of a Logic of Knowledge, Action, and Communication*, Ph.D. Thesis, New York University, forthcoming, 1987

Perlis, Don: *Language, Computation, and Reality*, unpublished manuscript, 1981

Sacerdoti, Earl: *A Structure for Plans and Behavior*, American Elsevier, New York 1977

Thomason, Richmond, ed: *Formal Philosophy: Selected Papers of Richard Montague*, Yale University Press, New Haven 1974

Thomason, Richmond: 'A Note on Syntactical Treatments of Modality,' *Synthese* 44, pp. 391-395, 1980

NYU COMPSCI TR-262 c.1
Morgenstern, Leora
Preliminary studies for a
first order logic of
knowledge and action

NYU COMPSCI TR-262 c.1
Morgenstern, Leora
Preliminary studies for a
first order logic of
knowledge and action

Mickey Berger
JAN 4 1989 *Mickey Berger*

FEB 1 1987

DATE DUE

DEC 0 3 1988

JAN 4 1989

